WEEK 9
SINGLE INHERITANCE

Dr. Yuzana Win

TOPICS COVERED

- Derived Class and Base Class
- Derived Class Constructors
- Overriding Member Functions
- Inheritance in the English Distance Class
- Class Hierarchies

WHAT IS INHERITANCE?

- Deriving a new class (subclass) from an existing class (base class or superclass).
- Inheritance creates a hierarchy of related classes (types) which share code and interface.

Another way

- Objects are often defined in terms of hierarchical classes with a base class and one or more levels of classes that inherit from the classes that are above it in the hierarchy.

WHY INHERITANCE ?

Inheritance is a mechanism for

- building class types from existing class types
- defining new class types to be a
  - specialization
  - augmentation

  of existing types

It means that Reuse of the same class and code

WHAT TO INHERIT?

- In principle, every member of a base class is inherited by a derived class
  - just with different access permission

- However, there are exceptions for
  - constructor and destructor
  - operator=() member
  - friends

  Since all these functions are class-specific

DEFINE A CLASS HIERARCHY

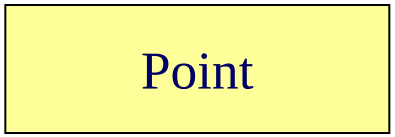
- Syntax:

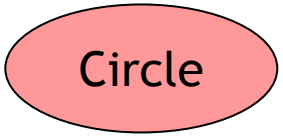
  class ***DerivedClassName*** **: access-level** ***BaseClassName***

  where
  - **access-level** specifies the type of derivation
    - **private** by default, or
    - **protected**
    - **public**

- Any class can serve as a base class
  - Thus a derived class can also be a base class

CLASSES WITHOUT INHERITANCE



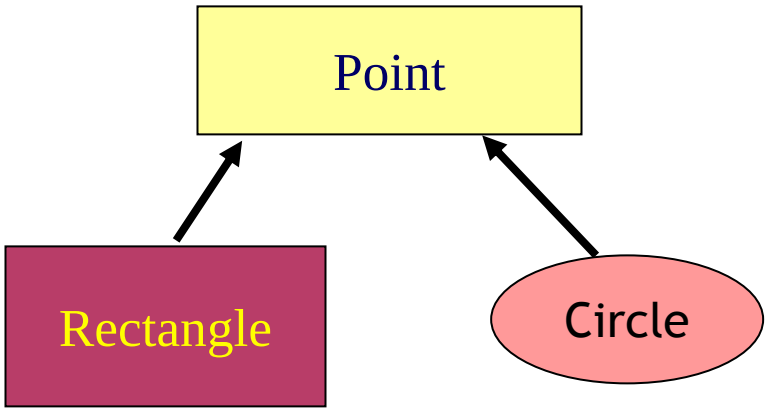



```
class Point{
        protected:
           int x,y;
        public:
           void set(int a, int b)
           { x=a; y=b; }
};
```

```
class Rectangle
{  private:
        int  width, length;
   public:
      void set(int a, int b, int w,
    int l)
     { x=a; y=b; width=w;
    length=l; }
};
```

```
class Circle{
        private:
           int x, y, radius;
        public:
           void set(int a, int b,
int r)
            { x=a; y=b; radius=r; }
};
```

CLASSES WITH INHERITANCE



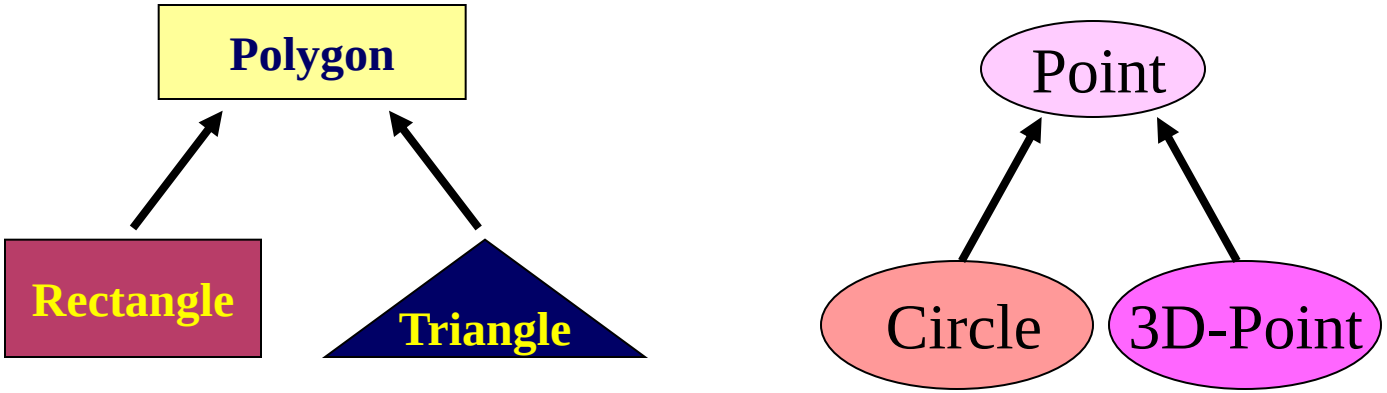

```
class Point
{   protected:
          int x,y;
     public:
          void set(int a, int b)
           { x=a; y=b; }
};
```

```
class Rectangle : public Point
{  private:
        int width, length;
   public:
      void set(int a, int b, int w,
    int l)
     { Point:: set(a,b);
        width=w; length=l;
     }
};
```

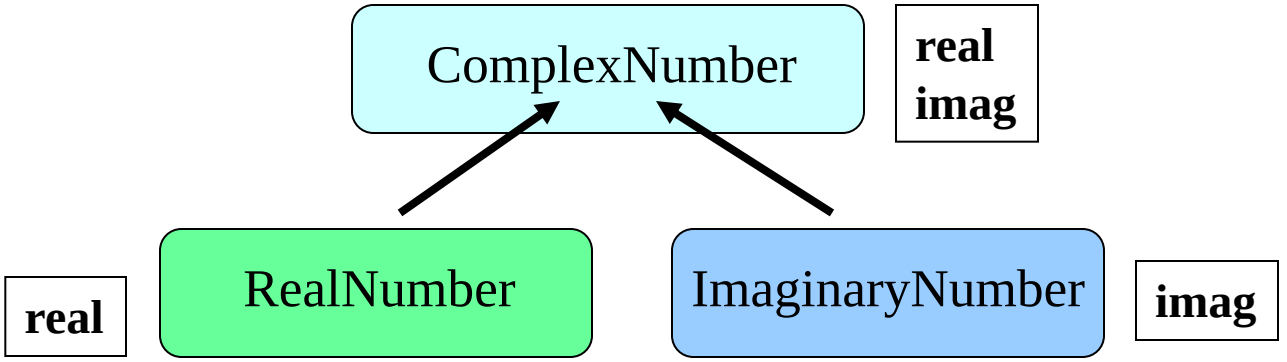
```
class Circle: public Point
{  private:
        int  x, y, radius;
    public:
        void set(int a, int b, int r)
          {  Point:: set(a,b);
              radius=r;
          }
    };
```

INHERITANCE CONCEPT

- Augmenting the original class



- Specializing the original class



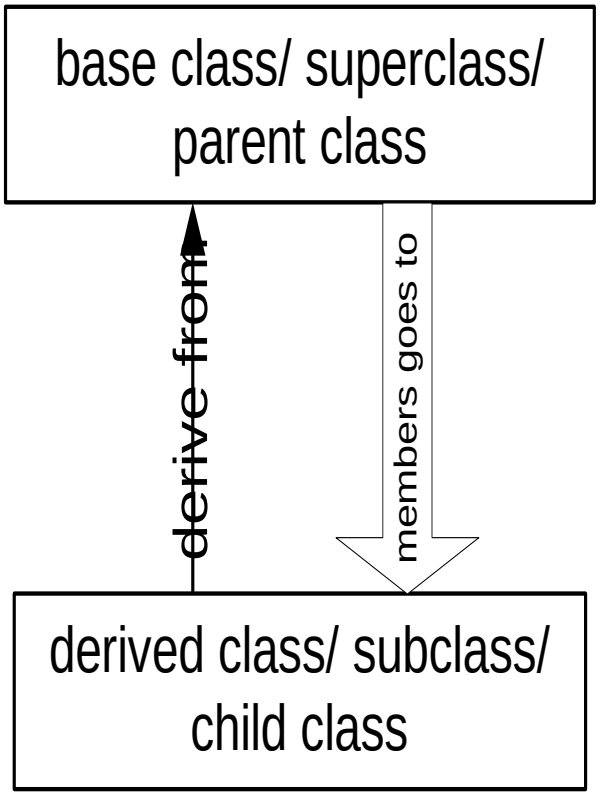
ACCESS RIGHTS OF DERIVED CLASSES

Type of Inheritance

Access Control for Members	private	protected	public
private	-	-	-
protected	private	protected	protected
public	private	protected	public

- The type of inheritance defines the minimum access level for the members of derived class that are inherited from the base class
- With public inheritance, the derived class follow the same access permission as in the base class
- With protected inheritance, only the public members inherited from the base class can be accessed in the derived class as protected members
- With private inheritance, none of the members of base class is accessible by the derived class

ACCESS CONTROL OVER THE MEMBERS

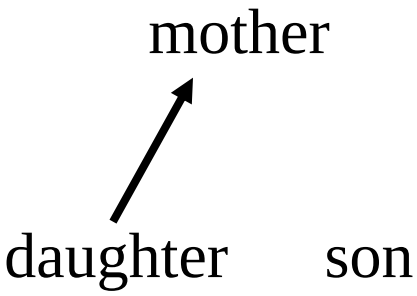


- Two levels of access control over class members
  - class definition
  - inheritance type

```
class Point{
    protected: int x, y;
    public: void set(int a, int b);
};
```

```
class Circle : public Point{
    … …
};
```

CLASS DERIVATION



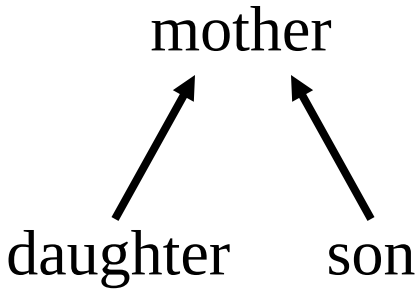

```
class daughter : public mother{
    private:
        double a;
    public:
        void foo ( );
};
```

```
class mother{
    protected:
        int x, y;
    public:
        void set(int a, int b);
    private:
        int z;
};
```

```
void daughter :: foo ( ){
    x = y = 20;
    set(5, 10);
    cout<<"value of a "<<a<<endl;
    z = 100;     // error, a private
    member
}
```

daughter can access 3 of the 4 inherited members

CLASS DERIVATION




```
class son : protected mother{
    private:
        double b;
    public:
        void foo ( );
};
```

```
class mother{
    protected:
        int x, y;
    public:
        void set(int a, int b);
    private:
        int z;
};
```

```
void son :: foo ( ){
    x = y = 20;
    set(5, 10);
    cout<<"value of b "<<b<<endl;
    z = 100;    // error, not a public
    member
}
```

son can access only 3 of the 4 inherited member

CONSTRUCTOR RULES FOR DERIVED CLASSES

The default constructor and the destructor of the base class are always called when a new object of a derived class is created or destroyed.

```
class A {
   public:
    A ( )
      {cout<< "A:default"<<endl;}
    A (int a)
    {cout<<"A:parameter"<<endl;}
};
```

```
class B : public A
{
   public:
    B (int a)
        {cout<<"B"<<endl;}
};
```

```
B test(1);
```

output:

```
A:default
B
```

CONSTRUCTOR RULES FOR DERIVED CLASSES

You can also specify an constructor of the base class other than the default constructor

```
DerivedClassCon (derivedClass args) :
BaseClassCon ( baseClass args )
{
    DerivedClass constructor body
}
```

```
class A {
  public:
    A ( )
     {cout<< "A:default"<<endl;}
    A (int a)
     {cout<<"A:parameter"<<endl;}
};
```

```
class C : public A
{
  public:
    C (int a):: A(a)
     {cout<<"C"<<endl;}
};
```

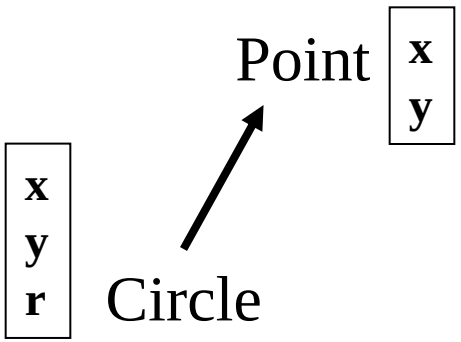
```
C test(1);
```

output:

```
A:parameter
C
```

DEFINE ITS OWN MEMBERS

The derived class can also define its own members, in addition to the members inherited from the base class




```
class Circle : public Point{
  private:
     double r;
  public:
      Point::set(a, b);
     void set_r( (double c);
};
```

```
class Point{
    protected:
       int x, y;
    public:
       void set(int a, int b);
};
```

Derived Members are:

```
protected:
       int x, y;  // from base
    private:
       double r;
    public:
       void set(int a, int b);   // from base
       void set_r(double c);
```

EVEN MORE …

- A derived class can override methods defined in its parent class. With overriding,
  - the method in the subclass has the identical signature to the method in the base class.
  - a subclass implements its own version of a base class method.

```
class A {
  protected:
    int x, y;
   public:
    void print ()
          {cout<<"From A"<<endl;}
};
```

```
class B : public A
{
   public:
     void print ()
         {cout<<"From B"<<endl;}
};
```

Access a Method

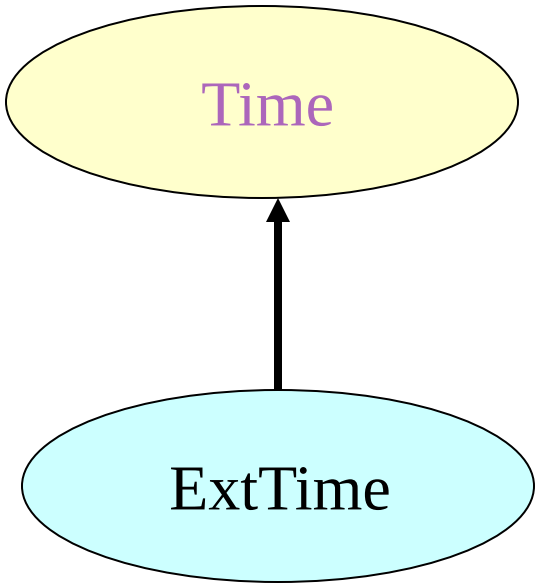
```
class Point
{
    protected:
       int x, y;
    public:
       void set (int a, int b)
           {x=a; y=b;}
       void foo ();
       void print();
};
```

```
class Circle : public Point{
  private:  double r;
  public:
   void set (int a, int b, double c)
   { Point :: set(a, b);//same name function call
      r = c;
   }
    void print();
};
```

```
Point A;
A.set(30,50);   // from base class
    Point
A.print(); // from base class Point
```

```
Circle C;
C.set(10,10,100);   // from class
Circle
C.foo ();   // from base class Point
C.print(); // from class Circle
```

Derived Class to Base



- Time is the base class
- ExtTime is the derived class with public inheritance
- The derived class can
  - inherit all members from the base class, except the constructor
  - access all public and protected members of the base class
  - define its private data member
  - provide its own constructor
  - define its public member functions
  - override functions inherited from the base class

Derived Class and Base Class

```
// counten.cpp
// inheritance with Counter class
#include <iostream>
using namespace std;
class Counter
{
protected:
        unsigned int count;
public:
        Counter() : count(0)
        { }
        Counter(int c) : count(c)
        { }
        unsigned int get_count() const
        { return count; }
        Counter operator ++ ()
        { return Counter(++count); }
};
class CountDn : public Counter
//derived class
{
public:
        Counter operator -- ()
                                        -count); }
```

```
c1=0
c1=3
c1=1
Press any key to continue . . .
```

```
        cout << "\nc1=" <<
        c1.get_count();
        ++c1; ++c1; ++c1;
        cout << "\nc1=" <<
        c1.get_count();
        --c1; --c1;
        cout << "\nc1=" <<
        c1.get_count();
        cout << endl;
        return 0;
}
```

Constructors, Destructors, and Inheritance

When Constructor and Destructor Functions Are Executed?

```
#include <iostream>
using namespace std;
class base {
 public:
  base() { cout << "Constructing
base\n"; }
  ~base() { cout << "Destructing
base\n"; }
};

class derived: public base {
public:
    derived() { cout << "Constructing
                    derived\n"; }
   ~derived() { cout << "Destructing
                     derived\n"; }
};
```

```
int main()
{
      derived ob;
      return 0;
}
```

output

```
Constructing base
Constructing derived
Destructing derived
Destructing base
```

Overriding Member Functions

```
// staken.cpp
// overloading functions in base and
derived classes
#include <iostream>
using namespace std;
#include <process.h> //for exit()
class Stack
{
protected:
        enum { MAX = 3 };
        int st[MAX];
        int top;
public:
        Stack()         //constructor
        { top = -1; }
        void push(int var)
        { st[++top] = var; }
        int pop()
        { return st[top--]; }
};
```

```
class Stack2 : public Stack
{
public:
        void push(int var)
{
     if(top >= MAX-1)
     { cout << "\nError: stack is
      full";    exit(1); }
       Stack::push(var);
}
       int pop() {
       if(top < 0)
       { cout << "\nError: stack is
       empty\n"; exit(1); }
       return Stack::pop();
}
};
```

Overriding Member Functions

```
int main()
{
   Stack2 s1;
   s1.push(11);       //push some values onto stack
   s1.push(22);
   s1.push(33);
   cout << endl << s1.pop();  //pop some values from stack
   cout << endl << s1.pop();
   cout << endl << s1.pop();
   cout << endl << s1.pop();    //oops, popped one too many...
   cout << endl;
   return 0;
}
```

Output

```
33
22
11
Error: stack is empty
```

Inheritance in the English Distance Class

```
// englen.cpp
// inheritance using English Distances
#include <iostream>
using namespace std;
enum posneg { pos, neg };
class Distance
{
protected:
        int feet;
        float inches;
public:
        Distance() : feet(0), inches(0.0)
        {  }
        Distance(int ft, float in) :
        feet(ft), inches(in)
        {  }
        void getdist() {
cout << "\nEnter feet: ";  cin >> feet;
cout << "Enter inches: ";  cin >> inches;
}
void showdist() const
{ cout << feet << "\'-" << inches
<< '\"'; }
};

class DistSign : public Distance
{
private:
        posneg sign;
public:
        DistSign() : Distance()
        { sign = pos; }
        DistSign(int ft, float in,
posneg sg=pos) : Distance(ft, in)
        { sign = sg; }
        void getdist()
        { Distance::getdist();
          char ch;
```

Inheritance in the English Distance Class

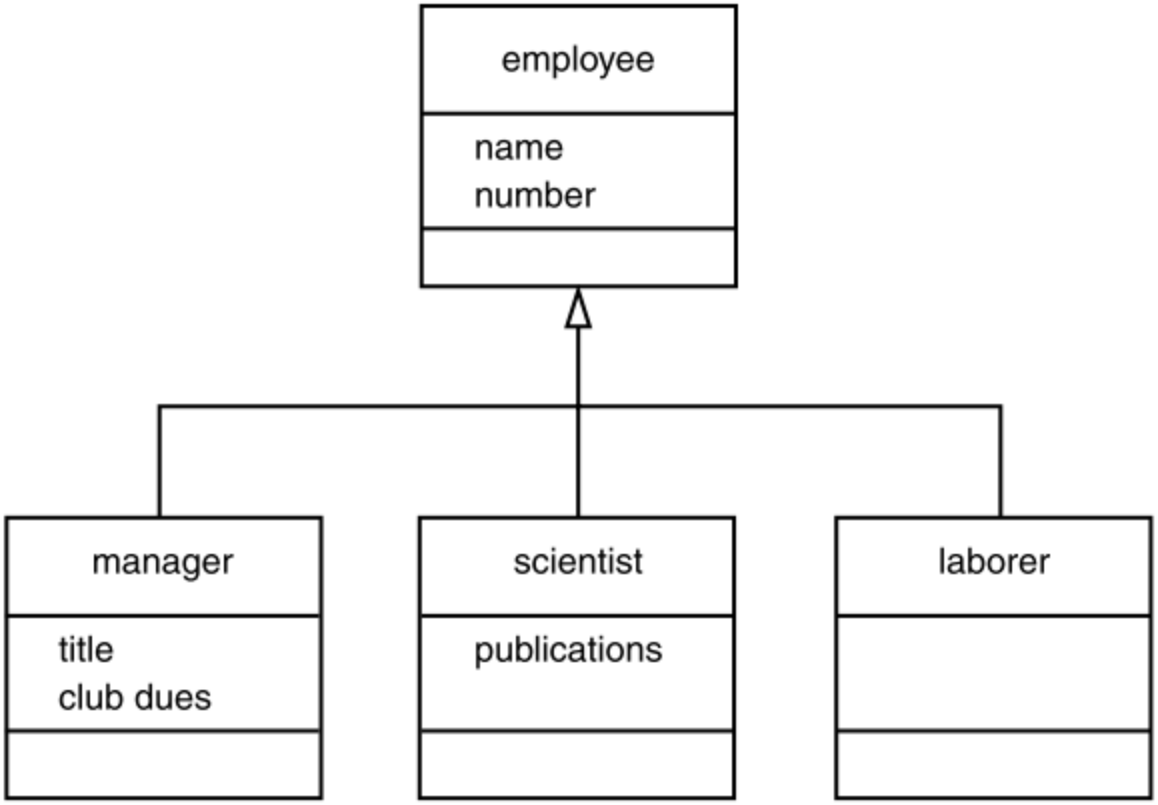
```
cout << “Enter sign (+ or -): ”;
cin >> ch;
sign = (ch==‘+’) ? pos : neg;
}
void showdist() const
{
   cout << ( (sign==pos) ? “(+)” : “(-)” );
Distance::showdist();
}
};
int main()
{
    DistSign alpha;
    alpha.getdist();
    DistSign beta(11, 6.25);
     DistSign gamma(100, 5.5, neg);
     //display all distances
```

```
cout << “\nalpha =”;
alpha.showdist();
cout << “\nbeta = ”;
beta.showdist();
cout << “\ngamma = ”;
gamma.showdist();
cout << endl;
return 0;
}
```

```
Enter feet:
6
Enter inches: 2.5
Enter sign (+ or -): -

alpha =(-)6'-2.5"
beta = (+)11'-6.25"
gamma = (-)100'-5.5"
Press any key to continue . . .
```

Class Hierarchies



UML Class Diagram for EMPLOYEE

Inheritance

```
// employ.cpp
// models employee database using inheritance
#include <iostream>
using namespace std;
const int LEN = 80;
class employee                    //employee class
{
private:
      char name[LEN];             //employee name
      unsigned long number;       //employee number
public:
      void getdata()
      { cout << "\n   Enter last name:";
        cin >> name;
        cout << " Enter number:";
        cin >> number; }
       void putdata() const
       { cout << "\n   Name: " << name;
         cout << "\n   Number: " << number;
       }};
```

```
class manager : public em
{
private:
    char title[LEN];
    double dues;
public:
     void getdata()
{ employee::getdata();
  cout << " Enter title:";
  cin >> title;
  cout << " Enter golf clu
   cin >> dues;
}
void putdata() const
{ employee::putdata();
  cout << "\n   Title:" <<
  cout << "\n Golf club du
dues;
}
};
```

Inheritance

```
class scientist : public employee
//scientist class
{
private:
          int pubs;
public:
         void getdata()
        { employee::getdata();
          cout << "Enter number of pubs:";
          cin >> pubs;
          }
          void putdata() const
          {
          employee::putdata();
          cout << "\n Number of
          publications: " << pubs;
          }};
class laborer : public employee
//laborer class
{};
```

```
int main()
{
  manager m1, m2;
  scientist s1;
  laborer l1;
  cout << endl;
  cout << "\nEnter data for
manager 1";
  m1.getdata();
  cout << "\nEnter data for
manager 2";
  m2.getdata();
  cout << "\nEnter data for
scientist 1";
   s1.getdata();
   cout << "\nEnter data for
laborer 1";
   l1.getdata();
```

Inheritance

```
//display data for several employees
cout << “\nData on manager 1”;
m1.putdata();
cout << “\nData on manager 2”;
m2.putdata();
cout << “\nData on scientist 1”;
s1.putdata();
cout << “\nData on laborer 1”;
l1.putdata();
cout << endl;
return 0;
}
```

```
Enter data for manager 1
   Enter last name: Wainsworth
 Enter number:10
 Enter title:President
 Enter golf club dues:1000000

Enter data for manager 2
   Enter last name:Bradley
 Enter number:124
 Enter title:Vice-President
 Enter golf club dues:500000

Enter data for scientist 1
   Enter last name: Frenglish
 Enter number:23456
Enter number of pubs:999

Enter data for laborer 1
   Enter last name:John
 Enter number:543246

Data on manager 1
   Name: Wainsworth
   Number: 10
   Title:President
 Golf club dues:1e+006
Data on manager 2
   Name: Bradley
   Number: 124
   Title:Vice-President
 Golf club dues:500000
Data on scientist 1
   Name: Frenglish
   Number: 23456
 Number of      publications: 999
Data on laborer 1
   Name: John
   Number: 543246
Press any key to continue . . .
```

ASSIGNMENT 1 : SINGLE INHERITANCE

- Create a base class, Person, which includes name, nrc, DOB, and show and get functions.
- Create a child class, Student, which includes rollno, paper1, paper2 and functions such as show, get, calculateResult.
- Create another child class, Teacher, which includes rank, dept and salary and functions such as show, get, promote and raise salary.
- Write a main() function and test the inheritance hierarchy.

ASSIGNMENT 2 : SINGLE INHERITANCE

- Imagine a publishing company that markets both Disk (Hard disk/ removable disk) and Printer (Laser Printer/ Inkjet Printer/ Dot Matrix Printer) of its works. Create a class called ***Product*** that stores name (a string) and price (type float) for this two market products. From this class derive two class: ***Disk***, which adds the storage_capacity (type int) to store storage capacity data of disk. The another class is ***Printer***, which adds the page_count(type int) to store the pages that this printer can print for one cartridge. Each of these three classes should have a getData() function to get its data from user at the keyboard, and showData() function to display the data.